Proceedings of the First International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2014)
Porto, Portugal

Jesus Carretero, Javier Garcia Blas
Jorge Barbosa, Ricardo Morla
(Editors)

August 27-28, 2014

Improving the Performance of the MPI_Allreduce Collective Operation through Rank Renaming

Juan-Antonio Rico-Gallego
University of Extremadura, Spain
jarico@unex.es

Juan-Carlos Díaz-Martín
University of Extremadura, Spain
juancarl@unex.es

Abstract

Collective operations, a key issue in the global efficiency of HPC applications, are optimized in current MPI libraries by choosing at runtime between a set of algorithms, based on platform-dependent beforehand established parameters, as the message size or the number of processes. However, with progressively more cores per node, the cost of a collective algorithm must be mainly imputed to process-to-processor mapping, because its decisive influence over the network traffic. Hierarchical design of collective algorithms pursuits to minimize the data movement through the slowest communication channels of the multi-core cluster. Nevertheless, the hierarchical implementation of some collectives becomes inefficient, and even impracticable, due to the operation definition itself. This paper proposes a new approach that departs from a frequently found regular mapping, either sequential or round-robin. While keeping the mapping, the rank assignation to the processes is temporarily changed prior to the execution of the collective algorithm. The new assignation makes the communication pattern to adapt to the communication channels hierarchy. We explore this technique for the Ring algorithm when used in the well-known MPI_Allreduce collective, and discuss the obtained performance results. Extensions to other algorithms and collective operations are proposed.

Keywords MPI Collectives, Parallel Algorithms, Message Passing Interface, Multi-core Clusters

I. Introduction

MPI [1] collective functions involve a group of processes communicating by message passing in an isolated context, known as *communicator*. Each process of a communicator is identified by its *rank*, an integer number ranging from 0 to $P-1$, where P is the size of the communicator. The optimisation of collectives is a key issue in HPC applications. A collective operation can be executed by different algorithms, each suitable for a given network technology, communicator size, message size, etc. For example, in the MPICH library [2], the implementation of *MPI_Allreduce* uses two algorithms for medium and large messages when the number of processes is a power of two, namely *Recursive Doubling* and *Ring*. The switch from the first to the second algorithm is done at execution time, with platform-dependent beforehand established message size and process number thresholds.

Current parallel systems are composed of multi-core nodes connected by a high performance network. The communication cost between two MPI ranks depends on their location, being lower if they share memory, and higher if they are in different nodes. Therefore the performance of an application depends on the assignation of the ranks to the processors of the cluster (mapping). In general, two types of mapping cover the necessities of most applications: *sequential* and *round-robin*. In the sequential mapping, ranks bind to processors so that a domain is completed (e.g. socket or node) before moving to the next domain. In round-robin, ranks are bound to domains by rotating on the existing domains.

Mapping affects to the performance of the underlying algorithms of collective operations. Interestingly, a given mapping may favour an algorithm and, at the same time, being harmful to another algorithm, not matter if both are used in the implementation of the same collective. For example, in the implementation of the *allreduce* operation in MPICH, referred above, the Recursive Doubling algorithm shows a better performance when the mapping is round-robin, while the Ring algorithm runs faster under the sequential mapping.

An approach to the issue of collectives performance is building algorithms that are aware of the different capacities of the available communication channels, as shared memory and network. These algorithms, known as *hierarchical*, stand on minimizing the communications through the slower channels, but the implementation for some collectives as *allgather* is not as effective as expected, even impracticable, and hence it is not provided in well-known MPI libraries as Open MPI [3].

This paper describes a new approach to the optimization of collectives in multi-core clusters. The goal is to obtain the best possible communication throughput. For instance, in the Ring algorithm, the communication takes place between consecutive ranks. If consecutive ranks are mapped to different nodes, all the communications progress through the network. Instead, a schedule of consecutive ranks to processes placed in the same multi-core node favours the much more efficient shared memory communication. Our method is based on a temporal reassignment of ranks. That neither modifies the algorithm nor the physical mapping. Instead, it is carried out by means of a transformation function prior to the execution of the algorithm. The function is simple and efficient, and converts a sequential mapping to round-robin and vice versa only during the execution of the algorithm.

This paper focuses on the *Ring* algorithm in the context of the *allreduce* operation. Besides, the methodology described is directly applicable to other algorithms used in the implementation of MPI

collectives. Platform considered is characterized by P, the number of processors (or processes involved in the operation), and M, the number of nodes in the cluster. $Q = P/M$ is the number of processors per node. Two channels are considered in the system, shared memory and network, with different performance. The study is conducted under two different mappings, sequential and round-robin, under the assumption of a homogeneously distributed number of processes over the nodes of the system. A hierarchical implementation of the algorithm is examined as well. The attained cost reduction depends on the number of nodes and the number of processes per node. In the used experimental platform, even with a small number of processes and nodes, the improvement reaches up to $2\times$ for long messages.

With respect to the structure of this article, following this introduction, section II reviews proposals of optimization of collective operations in a broad range of platforms. Section III studies the allreduce *Ring* algorithm in multi-core clusters based on the incoming mapping, and also a hierarchical allreduce implementation. The section exposes our proposal to improve the performance of the algorithm and the section IV outlines extensions to cases not covered in this paper. Section V shows the obtained performance figures, and section VI concludes the paper.

II. Related Work

MPI collectives performance is a key issue in high performance computing applications, and significant work has been invested in their design and optimization. Collectives in the MPI standard can be implemented from several of a set of algorithms available.

For instance, *MPI_Allreduce* can be implemented using the *Recursive Doubling* algorithm, that improves the latency when P is a power of two for small messages because is optimum with regard to the number of stages, however, the *Ring* algorithm performs better for larger messages. Both algorithms are also used in the implementation of *MPI_Allgather*, for which, in addition, other proposed algorithms improve the performance when requirements related to message size, process number or hardware and network technologies are met. *Bruck* algorithm [4] is more efficient for very short messages, even though it needs additional temporal memory, *Neighbour Exchange* algorithm in [5] requires half the stages than the Ring algorithm when the number of processes is even, and it exploits the piggy-backing feature of the TCP/IP protocols, as well as the *Dissemination* algorithm, proposed in [6], based on processes pairwise exchange of messages. Also related to the improvement of the performance by exploiting some networks capabilities, Mamidala et al. [7] evaluate the RDMA capacity for allowing concurrent direct memory access by the processes either in the same or different node of a multi-core cluster. Ma et al. [8] discuss the intra-node processes direct copy communication through shared memory by using the capacities of the operating system, and in [9] evaluate its impact in the collectives operations. Kielmann et al. [10] focus on the optimization of collective communications for clustered wide area systems.

The use of several algorithms in the same collective, based on system dependant beforehand established thresholds for message size and number of processes is shown by Thakur et al. in work [11] in a monoprocessor cluster of workstations. This approach has been adopted by the MPICH library, and it is available in the Open MPI library through its Modular Component Architecture [12]. Vadhiyar et al. [13] evaluate such improvement of performance through previous executed series of experiments conducted in an specific platform.

Multi-core clusters introduce a new actor in the scene. Performance becomes dependant on the effective use of the different communication channels. Hierarchical algorithms are specifically built to minimize the use of slower communication channels, and usually execute in several stages [14]. The process group splits in subgroups, with a local root per subgroup. Processes in a subgroup communicate through the faster communication channel, usually shared memory, hence, a subgroup is assigned to a node in the system. The application of these kind of algorithms to several implementations of the MPI standard and hardware platforms is extensively evaluated in [15], [16] and [17]. Based on analytical communication models, Karonis et al. [18] demonstrated the advantages of a multilevel topology-aware implementation of algorithms with respect to optimal *plain* algorithms. Sack and Gropp [19] show that a suboptimal algorithm in terms of inter-domain communications may produce lesser congestion that an optimal algorithm, and therefore to achieve a faster execution.

Former approximations adapt algorithms to the underlying communication capabilities. An inverse approach is to improve the performance through the calculation of the best layout of the processes over the processors of the cluster. Kravtsov et al. [20] define and propose an efficient solution to the *topology-aware co-allocation problem*, and Jeannot et al. proposes the *TreeMatch* algorithm in [21] applied to multi-core clusters. The challenge is optimally mapping the graph that defines the communication necessities of an application to the graph of the available resources. The solution can be applied to MPI collective operations, provided that they are built as a set of point-to-point transmissions [22]. Algorithms to automatically build the optimal *distance-aware collective communication topology*, based on the distance information between processes, are proposed in [23]. The results are applied to *Binomial Tree broadcast* and *Ring allgather* collectives.

III. MPI_Allreduce Ring Algorithm

In the *MPI_Allreduce* collective operation every process contributes with a buffer of size m bytes and gets in the output buffer the result of applying an specified operation to all the P processes buffers.

Ring algorithm implementation of the allreduce collective first copies data from the input buffer to the output buffer. Next, it operates on the output buffer in two phases: *computation* and *distribution*. The algorithm does not preserve order of operations. As a consequence, it can not be used with non commutative operations.

The *computation phase* is done in $P-1$ stages. The data buffer is divided up in segments of size m/P. In each stage k, from $k=0$, a process p sends its $p-k$ segment to process $p+1$, and next receives in a temporary buffer a segment from process $p-1$, that operates with local $p-k-1$ segment, with wraparounds. The operated segment in each process will be sent in the next stage. After $P-1$ stages, each process p has a full operated segment in the $p+1$ position of the output buffer.

Distribution phase performs an allgather to distribute these segments between processes also using a Ring algorithm. The algorithm operates in $P-1$ stages. All processes contribute with an m/P bytes segment at offset $p+1$ and receive P segments ordered by rank, for

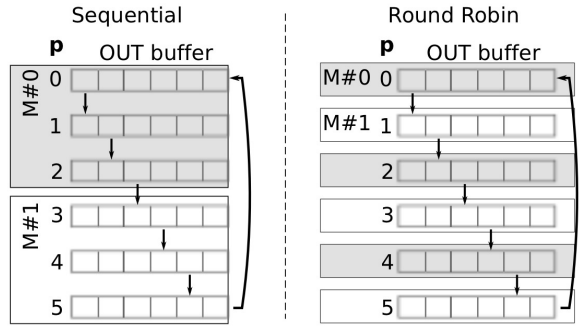

Figure 1: Representation of the communications in the stages of the Allreduce Ring (both computation and distribution phases) algorithm when processes are sequentially and round robin mapped, in a machine with $P = 6$ processes and $M = 2$ nodes.

a total of m bytes. In each stage, process p sends to process $p+1$ the m/P bytes received in the previous stage from process $p-1$, with wraparound.

Figure 1 represents the transmissions between processes in a machine with $P = 6$ processes and $M = 2$ nodes under both sequential (left) and round robin (right) mapping. In the Ring algorithm sequential mapping minimizes the point-to-point transmissions across the network. The first process of each node receives from the former node, and the last process sends to the next node. The rest of transmissions take place in shared memory and they progress in parallel with a total of M inter-node flows. Nevertheless, when the incoming mapping is round-robin $M \times Q$ inter-node transmissions take place at a time, giving rise to a much higher contention that degrades the communication. In networks as Ethernet, for instance, the contention may lead to a performance breakdown that grows with the number of simultaneous transfers. In section V, we evaluate the performance of Ring algorithm with sequential and round-robin mappings in a cluster based on Infiniband.

The fact is that the layout of the processes over the processors has a great impact in the effective cost. In the allreduce, both phases, computation and distribution, communicate each rank with the nearest next and previous rank numbers, hence, consecutive ranks must run in the same node in order to increase the data transmissions inside a node and minimize the network contention. The next section explores the design of algorithms which take into account the mapping of processes on the physical hierarchy of communications.

III.1 Hierarchical Allreduce Algorithm

An algorithm can be designed to minimize the data movement through the slowest communication channels in a multi-core cluster for the allreduce collective. For example, the implementation found in *Open MPI* library is composed of three phases, that progress sequentially.

The first phase performs a *reduce* operation local to each node in the system. One process per node, called local root, obtains the full operated Q segments in the output buffer. The second phase performs an inter-node allreduce between local roots. The number of processes running allreduce decreases with respect to a simple allreduce operation in section III, from P to M, but the size of messages contributed by each process increases from m/P to $m \cdot Q/P$. Thus, the amount of data transmitted through the network is the same as the allreduce Ring algorithm with sequential mapping. Nevertheless, this hierarchical design of the allgather minimizes the network contention regardless of the initial process mapping. Finally, in the third phase, the local root process broadcasts its resulting buffer to the rest of the processes in the same node.

Additional communicators must be created to perform collectives inside each node, and the inter-node allreduce between local roots.

III.2 Allreduce Mapping Transformation at Run-Time

Under awareness of a regular mapping, such as sequential or round-robin, the programmer would be in the position of exploiting that knowledge to increase the algorithm performance.

Necessity of minimize network communication advises a physical rearrangement of the processes that guarantees a sequential mapping before starting Ring algorithm. In practice, however, physically moving the processes conveys an excessive latency and cached data invalidation penalties. We propose instead a mere previous logical rearrangement of the ranks, a solution that is applied dynamically and efficiently. Logical renaming of processes ranks can be applied to both computation and distribution phases. The new algorithm is denoted as Ring*.

Let be a rank set $R = \{r_0, r_1, \ldots, r_{P-1}\}$ assigned to the P processes of a communicator following a Round Robin mapping. For simplicity and without loss of generality, we define $r_p = p$. The set R can be transformed into another set $S = \{s_0, s_1, \ldots, s_{P-1}\}$, which shows a sequential mapping, with $s_p = f_{SQ}(p)$. The *transformation function* f_{SQ} is defined as:

$$f_{SQ}(p) = ((p \times Q)\,\%P) + \lfloor p \,/\, M \rfloor \tag{1}$$

The number of nodes M must be known and the processes must be homogeneously distributed between nodes, i.e., the number of processes per node (Q) must be constant. See section IV for explanations about extensions to irregular mappings.

Similarly, a rank set $S = \{s_0, s_1, \ldots, s_{P-1}\}$ with $s_p = p$ and a sequential mapping can be transformed into a set $R = \{r_0, r_1, \ldots, r_{P-1}\}$, which shows a round robin mapping, with $r_p = f_{RR}(p)$. The transformation function f_{RR} (inverse of f_{SQ}) is defined as:

$$f_{RR}(p) = ((p \times M)\,\%P) + \lfloor p \,/\, Q \rfloor \tag{2}$$

In the allreduce computation phase, renaming of processes is applied prior to the execution of the Ring algorithm. A process with rank p behaves as a process with rank $f_{SQ}(p)$, applying the definition in (1). Then, in the stage k, a process sends the segment $f_{SQ}(p) - k$ to process behaving as $f_{SQ}(p)+1$, calculated as $f_{RR}\left(f_{SQ}(p)+1\right)$. Next, it receives a segment from process with rank $f_{RR}\left(f_{SQ}(p)-1\right)$ in a temporary buffer, that operates with local segment $f_{SQ}(p) - k - 1$.

For instance, process $p = 2$ in Figure 2 behaves as $f_{SQ}(2) = 1$. In the first stage $k = 0$, it sends the segment $f_{SQ}(2) - k = 1$ to process behaving as the rank $f_{SQ}(2)+1$, which is calculated as $f_{RR}\left(f_{SQ}(2)+1\right) = f_{RR}(2) = 4$, and receives from

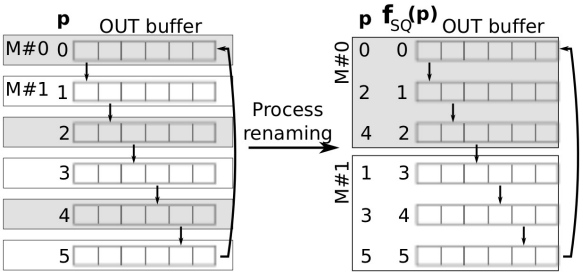

Figure 2: Allreduce Ring algorithm computation phase, stage $k = 0$, with round-robin mapping, $P = 6$ and $M = 2$. The buffer total size is divided up in P segments. Processes renaming through the transformation functions changes the mapping from round-robin to sequential before starting.

$f_{RR}\left(f_{SQ}(2) - 1\right) = f_{RR}(0) = 0$ a segment to be operated with the local segment number $f_{SQ}(2) - k - 1 = 0$.

Again, allreduce distribution phase requires renaming of processes from p into $f_{SQ}(p)$ prior to the execution of the regular Ring allgather algorithm. The operating principle is the same as the computation phase.

IV. Extensions of the Method

Our approach consists of departing from the a priori knowledge of a layout with regular mapping and keeps the original algorithm after having switched to another regular mapping, much more favourable in terms of performance. Such mapping information could be available through the processes manager module of the particular MPI implementation.

The transformation functions can be applied to algorithms with similar communication patterns to the Ring algorithm. For instance, *Neighbour Exchange* and *Binomial Tree* perform better when processes are sequentially mapped. Other algorithms have opposite requirements, such as *Recursive Doubling* and *Dissemination* algorithms, better suited to initial round robin mapping, because the distance between rank numbers communicating exponentially grows in each stage. The mapping needs to change from sequential to round robin, through the inverse application of the transformation functions.

The above-mentioned algorithms are used in a wide variety of collectives operations defined in the MPI standard, as *Broadcast*, *Scatter*, *Allgather*, etc. proving the method as highly generic.

Nevertheless, we can not always make assumptions about the deployment of the ranks over the cluster, all the more so as this layout may change with the creation of new communicators at run time, that could assign different ranks to the processes. On that case, with non-regular mappings, each rank involved in the collective operation will need to have information about the layout of all the ranks in the communicator. Resource requirements for that information are under study by the authors.

Performance measurements in clusters with other network technologies, such as Ethernet, confirms the expected results, with an increase in the difference of performance between mappings that is proportional to the difference in bandwidth capabilities between the channels.

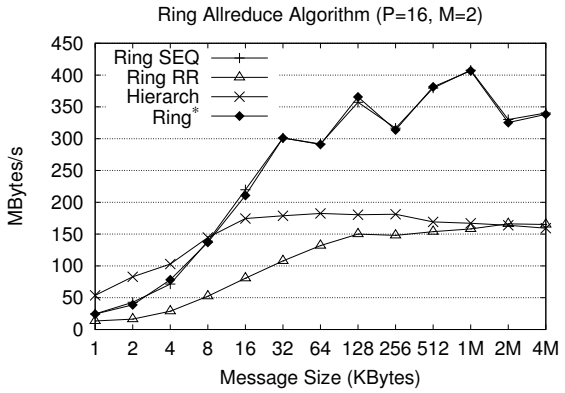

Figure 3: Bandwidth of the allreduce Ring algorithm with sequential and round-robin mappings, and the Ring algorithm when executed with $M = 2$ nodes and $Q = 8$ processes by node, for a total of $P = 16$ processes.*

V. Performance Evaluation

The experimental platform used, named *Fermi*, is composed of eight nodes connected by a QDR Infiniband network. Each node has two 2.27 GHz Quad core Intel Xeon E5520 processors, with 8MB of shared L3 cache size, making a total of eight cores per node. The operating system is Linux 2.6.32. We used IMB (Intel MPI Benchmark), version 3.2, to obtain the latency data. Bandwidth is calculated as the message size divided by the latency, and showed in figures for the sake of clarity. A high number of iterations are executed for each collective algorithm and mean time is taken. IMB runs on Open MPI 1.8, the library that provides the allreduce algorithms through its *Tuned* and *Hierarch* collective components. Nonetheless it should be noted that MVAPICH2 yields similar results, as well as MPICH, which has been tested in Ethernet networks.

The allreduce Ring algorithm performance is plotted in Figures 3 to 5. Figures represent the bandwidth of sequential (SEQ) and round robin (RR) mappings, as well as the *Ring** algorithm and the hierarchical implementation of the collective operation, for increasing number of nodes (M), with $Q = 8$. The difference in bandwidth between sequential and *Ring** algorithm with respect to less favourable round robin mapping is nearly to $2\times$, for all the range of messages. *Ring** overload to the Ring algorithm is very low, because it is only attributable to the execution of transformation functions.

Hierarchical implementation of the allreduce algorithm leads to a higher performance than round robin, but it degrades with the size of the message because phases must progress sequentially. Performance depends as well on the algorithms used in each phase. In this paper we use the *binomial tree* algorithm for the *Reduce* and *Broadcast* algorithms in the phases 1 and 3, and allreduce *Ring* algorithm in the inter-node phase 2. This configuration outperforms even the allreduce Ring algorithm for short and medium messages.

In Figure 6 we plot the relative mean bandwidth (measured along the whole range of messages plotted in figures) between different cases for a constant number of nodes ($M = 8$), and a growing number of processes per node. Note that the difference between the sequential and the round-robin mapping grows with Q. As expected,

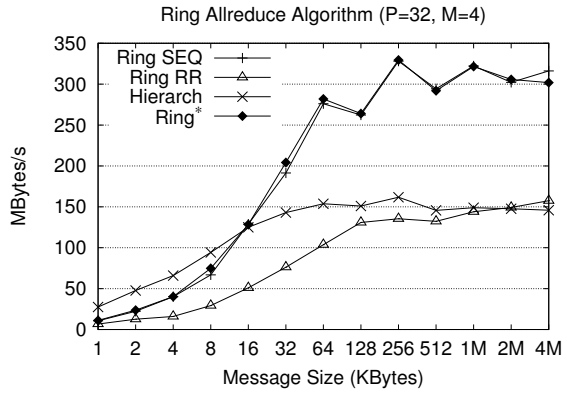

Figure 4: Bandwidth of the allreduce Ring algorithm with sequential and round-robin mappings, and the Ring algorithm when executed with $M = 4$ nodes and $Q = 8$ processes by node, for a total of $P = 32$ processes.*

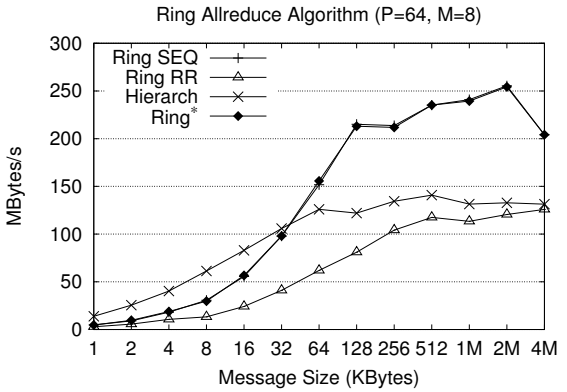

Figure 5: Bandwidth of the allreduce Ring algorithm with sequential and round-robin mappings, and the Ring algorithm when executed with $M = 8$ nodes and $Q = 8$ processes by node, for a total of $P = 64$ processes.*

the difference between sequential and *Ring** remains constant, and denotes a minimal overload. Respect to the hierarchical case, the difference is constant for that small number of nodes.

It can be observed in Figure 3 that the change of mapping of the *Ring** algorithm shows an improvement with respect to the round robin mapping even in a minimal configuration with only $M = 2$ nodes.

VI. Conclusions

The performance of MPI collective algorithms in multi-core clusters highly depends on the deployment of the processes on the processors of the system. These algorithms usually establish a communication pattern between ranks that, if under specific regular mappings, use the communication resources effectively, other mappings significantly worsen their performance. The hierarchical design pursues the optimal use of the system available communication channels, regardless of the process mapping, but they are only efficient in a limited subset of collectives operations.

This paper proposes a more generic approach, whose goal is to adapt the mapping of processes to the communication pattern of the collective algorithm in run-time to reduce network traffic and contention. Such a switch does not require process migration, but a renaming of the processes ranks prior to the execution of the original algorithm.

Performance improvements of *MPI_Allreduce* collective is evaluated when built upon the *Ring* algorithm, which performs better when processes are mapped sequentially. The figures show that the processes renaming adds a low impact upon the cost of the original algorithm. Results are also compared to the hierarchical implementation of the collective.

Our approach can be applied to other algorithms commonly used in MPI collective operations, as the Recursive Doubling, Neighbour Exchange, Dissemination or Binomial Tree, with different incoming mapping necessities, covering a broad range of communication patterns. In addition, the paper discusses extensions to cover non-regular mapping of processes and other collective operations.

Acknowledgment

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)', and by the computing facilities of Extremadura Research Centre for Advanced Technologies (CETA-CIEMAT), funded by the European Regional Development Fund (ERDF). CETA-CIEMAT belongs to CIEMAT and the Government of Spain.

References

[1] MPI Forum. MPI: A Message-Passing Interface Standard, Version 3.0., September 2012.

[2] MPICH. MPICH High Performance Portable MPI Implementation, 2013.

[3] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[4] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 8(11):1143–1156, 1997.

[5] Jing Chen, Linbo Zhang, Yunquan Zhang, and Wei Yuan. Performance evaluation of allgather algorithms on terascale linux cluster with fast ethernet. In *Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, HPCASIA '05, pages 437–, Washington, DC, USA, 2005. IEEE Computer Society.

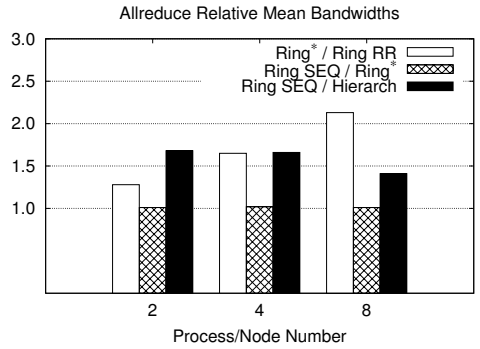

Figure 6: Allreduce relative mean bandwidths (calculated for the whole range of messages) of the Ring algorithm with respect to the Ring with round robin mapping, of Ring with sequential mapping with respect to Ring*, and of Ring* with respect to the hierarchical implementation. All the tests are executed with $M = 8$ nodes and $Q = 8$ processes by node, for a total of $P = 64$ processes.*

[6] Gregory D. Benson, Cho wai Chu, Qing Huang, and Sadik G. Caglar. A comparison of mpich allgather algorithms on switched networks. In *In Proceedings of the 10th EuroPVM/MPI 2003 Conference*, pages 335–343. Springer, 2003.

[7] Amith R. Mamidala, Abhinav Vishnu, and Dhabaleswar K. Panda. Efficient shared memory and rdma based design for mpi_allgather over infiniband. In *Proceedings of the 13th European PVM/MPI User's Group conference on Recent advances in parallel virtual machine and message passing interface*, EuroPVM/MPI'06, pages 66–75, Berlin, Heidelberg, 2006. Springer-Verlag.

[8] Teng Ma, G. Bosilca, A. Bouteiller, and J.J. Dongarra. Hierknem: An adaptive framework for kernel-assisted and topology-aware collective communications on many-core clusters. In *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 970–982, 2012.

[9] Teng Ma, G. Bosilca, A. Bouteiller, B. Goglin, J.M. Squyres, and J.J. Dongarra. Kernel assisted collective intra-node mpi communication among multi-core and many-core cpus. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 532–541, Sept 2011.

[10] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. Magpie: Mpi's collective communication operations for clustered wide area systems. *SIGPLAN Not.*, 34(8):131–140, May 1999.

[11] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in mpich. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.

[12] Jeffrey M. Squyres and Andrew Lumsdaine. The component architecture of open MPI: Enabling third-party collective algorithms. In Vladimir Getov and Thilo Kielmann, editors, *Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications*, pages 167–185, St. Malo, France, July 2004. Springer.

[13] S.S. Vadhiyar, G.E. Fagg, and J. Dongarra. Automatically tuned collective communications. In *Supercomputing, ACM/IEEE 2000 Conference*, pages 3–3, Nov 2000.

[14] Meng-Shiou Wu, R.A. Kendall, and K. Wright. Optimizing collective communications on smp clusters. In *Parallel Processing, 2005. ICPP 2005. International Conference on*, pages 399–407, 2005.

[15] Hao Zhu, David Goodell, William Gropp, and Rajeev Thakur. Hierarchical collectives in mpich2. In *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 325–326, Berlin, Heidelberg, 2009. Springer-Verlag.

[16] A.R. Mamidala, R. Kumar, D. De, and D.K. Panda. Mpi collectives on modern multicore clusters: Performance optimizations and communication characteristics. In *Cluster Computing and the Grid, 2008. CCGRID '08. 8th IEEE International Symposium on*, pages 130–137, 2008.

[17] Richard L. Graham and Galen Shipman. Mpi support for multicore architectures: Optimized shared memory collectives. In *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 130–140, Berlin, Heidelberg, 2008. Springer-Verlag.

[18] Nicholas T. Karonis, Bronis R. de Supinski, Ian T. Foster, William Gropp, and Ewing L. Lusk. A multilevel approach to topology-aware collective operations in computational grids. *CoRR*, cs.DC/0206038, 2002.

[19] Paul Sack and William Gropp. Faster topology-aware collective algorithms through non-minimal communication. *SIGPLAN Not.*, 47(8):45–54, February 2012.

[20] Valentin Kravtsov, Martin Swain, Uri Dubin, Werner Dubitzky, and Assaf Schuster. A fast and efficient algorithm for topology-aware coallocation. In *Proceedings of the 8th international conference on Computational Science, Part I*, ICCS '08, pages 274–283, Berlin, Heidelberg, 2008. Springer-Verlag.

[21] E. Jeannot, G. Mercier, and F. Tessier. Process placement in multicore clusters:algorithmic issues and practical techniques. *Parallel and Distributed Systems, IEEE Transactions on*, 25(4):993–1002, April 2014.

[22] Jin Zhang, Jidong Zhai, Wenguang Chen, and Weimin Zheng. Process mapping for mpi collective communications. In *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par '09, pages 81–92, Berlin, Heidelberg, 2009. Springer-Verlag.

[23] Teng Ma, T. Herault, G. Bosilca, and J.J. Dongarra. Process distance-aware adaptive mpi collective communications. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 196–204, 2011.